Phylogenetic Networks Do not Need to Be Complex: Using Fewer Reticulations to Represent Conflicting Clusters

Leo van Iersel[1], Steven Kelk[2], Regula Rupp[3] and Daniel Huson[3]

[1] University of Canterbury, Department of Mathematics and Statistics,
Private Bag 4800, Christchurch, New Zealand. l.j.j.v.iersel@gmail.com
[2] Centrum voor Wiskunde en Informatica (CWI)
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. s.m.kelk@cwi.nl
[3] Center for Bioinformatics ZBIT, Tübingen University
Sand 14, 72076 Tübingen, Germany. {huson,rrupp}@informatik.uni-tuebingen.de

Abstract Phylogenetic trees are widely used to display estimates of how groups of species evolved. Each phylogenetic tree can be seen as a collection of *clusters*, subgroups of the species that evolved from a common ancestor. When phylogenetic trees are obtained for several data sets (e.g. for different genes), then their clusters are often contradicting. Consequently, the set of all clusters of such a data set cannot be combined into a single phylogenetic tree. *Phylogenetic networks* are a generalization of phylogenetic trees that can be used to display more complex evolutionary histories, including *reticulate events* such as hybridizations, recombinations and horizontal gene transfers. Here we present the new CASS algorithm that can combine any set of clusters into a phylogenetic network. We show that the networks constructed by CASS are usually simpler than networks constructed by other available methods. Moreover, we show that CASS is guaranteed to produce a network with at most two reticulations per biconnected component, whenever such a network exists. We have implemented CASS and integrated it in the freely available Dendroscope software.

1 Introduction

Phylogenetics studies the reconstruction of evolutionary histories from genetic data of currently living organisms. A (rooted) *phylogenetic tree* is a representation of such an evolutionary history in which species evolve by mutation and speciation. The leaves of the tree represent the species under consideration and the root of the tree represents their most recent common ancestor. Each internal node represents a speciation: one species splits into several new species. Thus, mathematically speaking, such a node has indegree one and outdegree at least two. In recent years, a lot of work has been done on developing methods for computing (rooted) *phylogenetic networks* [5,10], which form a generalization of phylogenetic trees. Next to nodes representing speciation, rooted phylogenetic networks can also contain *reticulations*: nodes with indegree at least two. Such nodes can be used to represent recombinations, hybridizations or horizontal gene transfers, depending on the biological context. In addition, phylogenetic networks can also be interpreted in a more abstract sense, as a visualization of contradictory phylogenetic information in a single diagram.

Suppose we wish to investigate the evolution of a set $\mathcal{X}$ of taxa (e.g. species or strains). Each edge of a rooted phylogenetic tree represents a *cluster*: a proper subset of the taxon set $\mathcal{X}$. In more detail, an edge (u, v) represents the cluster containing those taxa that are descendants of v. Each phylogenetic tree T is uniquely defined by the set of clusters represented by T. Phylogenetic networks also represent clusters. Each of their edges represents one "hardwired" and at least one "softwired" cluster. An edge (u, v) of a phylogenetic network *represents* a cluster $C \subset \mathcal{X}$ *in the hardwired sense* if C equals the set of taxa that are descendants of v. Furthermore, (u, v) *represents* C *in the softwired sense* if C equals the set of all taxa that can be reached from v when, for each reticulation r, exactly one incoming edge of r is "switched on" and the other incoming edges of r are "switched off". An equivalent definition states that a phylogenetic network N *represents* a cluster C *in the softwired sense* if there exists a tree T that is displayed by N (formally defined below) and represents C. In this paper we will always use "represent" in the softwired sense. It is usually the

clusters in a tree that are of more interest, and less the actual trees themselves, as clusters represent putative monophyletic groups of related species. For a complete introduction to clusters see Huson and Rupp [10].

In phylogenetic analysis, it is common to compute phylogenetic trees for more than one data set. For example, a phylogenetic tree can be constructed for each gene separately, or several phylogenetic trees can be constructed using different methods. To accurately reconstruct the evolutionary history of all considered taxa, one would preferably like to use the set $\mathcal{C}$ of all clusters represented by at least one of the constructed phylogenetic trees. In general however, some of the clusters of the different trees will be conflicting, which means that there will be no single phylogenetic tree representing $\mathcal{C}$. Therefore, several recent publications have studied the construction of a phylogenetic *network* representing $\mathcal{C}$. Huson and Rupp [11] describe how a phylogenetic network can be constructed that represents $\mathcal{C}$ in the hardwired sense (a *cluster network*). A network is a *galled network* if it contains no path between two reticulations that is contained in a single *biconnected component* (a maximal subgraph that cannot be disconnected by removing a single node). Huson and Klöpper [8] and Huson et al. [12] describe an algorithm for constructing a galled network representing $\mathcal{C}$ in the softwired sense.

Related literature describes the construction of phylogenetic networks from phylogenetic trees or *triplets* (phylogenetic trees on three taxa). A tree or triplet T is *displayed* by a network N if there is a subgraph T' of N that is a subdivision of T (i.e. T' can be obtained from T by replacing edges by directed paths). Computing the minimum number of reticulations required in a phylogenetic network displaying two input trees (on the same set of taxa) was shown to be APX-hard by Bordewich and Semple [2]. Bordewich et al. [1] proposed an exact exponential-time algorithm for this problem and Linz and Semple [18] showed that it is fixed parameter tractable (FPT), if parameterized by the minimum number of reticulations. The downside of these algorithms is that they are very rigid in the sense that one generally needs very complex networks in order to display the given trees.

The *level* of a binary network is the maximum number of reticulations in a biconnected component[4], and thus provides a measure of network complexity. Given an arbitrary number of trees on the same set of taxa, Huynh et al. [13] describe a polynomial-time algorithm that constructs a level-1 phylogenetic network that displays all trees and has a minimum number of reticulations, if such a network exists (which is unlikely in practice). Given a triplet for each combination of three taxa, Jansson, Sung and Nguyen [16,17] give a polynomial-time algorithm that constructs a level-1 network displaying all triplets, if such a network exists. The algorithm by van Iersel and Kelk [15] can be used to find such a network that also minimizes the number of reticulations. These results have later been extended to level-2 [14,15] and more recently to level-k, for all $k \in \mathbb{N}$ [19]. Although this work on triplets is theoretically interesting, it has the practical drawback that biologists are not interested in triplets (but rather in trees or clusters) and that these algorithms need a triplet for each combination of three taxa as input, while some triplets might be difficult to derive correctly.

In this article, we present the algorithm CASS[5], which takes any set $\mathcal{C}$ of clusters as input and constructs a phylogenetic network that represents $\mathcal{C}$. Furthermore, the algorithm aims at minimizing the level of the constructed network and in this sense CASS is the first algorithm to combine the flexibility of clusters with the power of level minimization. CASS constructs a phylogenetic tree representing $\mathcal{C}$ whenever such a tree exists. Moreover, we prove that CASS constructs a level-1 or level-2 network representing $\mathcal{C}$ whenever there exists a level-1 or level-2 network representing $\mathcal{C}$, respectively. Experimental results show that also when no level-2 network representing $\mathcal{C}$ exists, CASS usually constructs a network with a significantly lower level and lower number of reticulations compared to other algorithms. In fact, we conjecture that similar arguments as in our proof for level-2 can be used to show that CASS always constructs a level-k network with minimum k. We prove a decomposition theorem for level-k networks that supports this conjecture. Finally, we prove that CASS runs in polynomial time if the level of the output network is bounded by a constant.

We have implemented CASS and added it to our popular tree-drawing program Dendroscope [9], where it can be used as an alternative for the cluster network [11] and galled network [12] algorithms. Experiments show that, although CASS needs more time than these other algorithms, it constructs a simpler network representing the same set of clusters. For example, Figure 1(a) shows a set of clusters and the galled network

---

[4] In Section 2 we generalize the notion of *level* to non-binary networks.
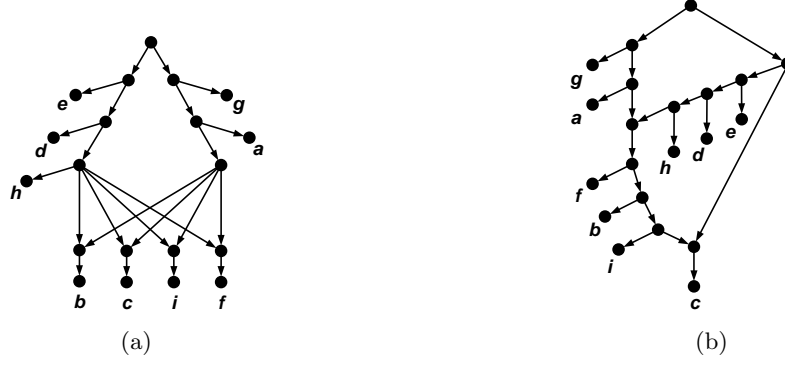[5] Named after the Cass Field Station in New Zealand.

Figure 1. (a) The output of the galled network algorithm [12] for $\mathcal{C} = \{\{a, b, f, g, i\}, \{a, b, c, f, g, i\}, \{a, b, f, i\},$ $\{b, c, f, i\}, \{c, d, e, h\}, \{d, e, h\}, \{b, c, f, h, i\}, \{b, c, d, f, h, i\}, \{b, c, i\}, \{a, g\}, \{b, i\}, \{c, i\}, \{d, h\}\}$ and (b) the network constructed by CASS for the same input.

with four reticulations constructed by the algorithm in [12]. However, for this data set also a level-2 network with two reticulations exists, and CASS can be used to find this network, see Figure 1(b). Dendroscope now combines the powers of CASS and the two previously existing algorithms for constructing galled- and cluster networks.

2 Level-k Networks and Clusters

Consider a set $\mathcal{X}$ of taxa. A *(phylogenetic) network* (on $\mathcal{X}$) is a directed acyclic graph with a single root and leaves bijectively labeled by $\mathcal{X}$. The indegree of a node v is denoted $\delta^-(v)$ and v is called a *reticulation* if $\delta^-(v) \geq 2$. An edge (u, v) is called a *reticulation edge* if its head v is a reticulation and is called a *tree edge* otherwise. The *reticulation number* of a phylogenetic network $N = (V, E)$ is defined as

$$\sum_{v \in V : \delta^-(v) > 0} (\delta^-(v) - 1) = |E| - |V| + 1 \ .$$

A directed acyclic graph is *connected* (also called "weakly connected") if there is an undirected path (ignoring edge orientations) between each pair of nodes. A node (edge) of a directed graph is called a *cut-node* (*cut-edge*) if its removal disconnects the graph. A directed graph is *biconnected* if it contains no cut-nodes. A biconnected subgraph B of a directed graph G is said to be a *biconnected component* if there is no biconnected subgraph $B' \neq B$ of G that contains B.

A phylogenetic network is said to be a *level-k network* if each biconnected component has reticulation number at most k.[6] A phylogenetic network is called *binary* if each node has either indegree at most one and outdegree at most two or indegree at most two and outdegree at most one. Note that the above definition of level generalizes the original definition [3] for binary networks. A level-k network is called a *simple level-$\leq k$ network* if the head of each cut-edge is a leaf. A simple level-$\leq k$ network is called a *simple level-k network* if its reticulation number is precisely k. A *phylogenetic tree* (on $\mathcal{X}$) is a phylogenetic network (on $\mathcal{X}$) without reticulations, i.e. a level-0 network.

Consider a set of taxa $\mathcal{X}$. Proper subsets of $\mathcal{X}$ are called *clusters*. We say that two clusters $C_1, C_2 \subset \mathcal{X}$ are *compatible* if either $C_1 \cap C_2 = \emptyset$ or $C_1 \subset C_2$ or $C_2 \subset C_1$. Consider a set of clusters $\mathcal{C}$. We say that a set of taxa $X \subset \mathcal{X}$ is *separated* (by $\mathcal{C}$) if there exists a cluster $C \in \mathcal{C}$ that is incompatible with X. The *incompatibility graph* $IG(\mathcal{C})$ of $\mathcal{C}$ is the undirected graph (V, E) that has node set $V = \mathcal{C}$ and edge set

$$E = \{\{C_1, C_2\} \mid C_1 \text{ and } C_2 \text{ are incompatible clusters in } \mathcal{C}\} \ .$$

---

[6] Note that to determine the reticulation number of a biconnected component one only counts edges inside this biconnected component.

3 Decomposing Level-k Networks

In this section, we describe the general outline of our algorithm CASS. We show how the problem of determining a level-k network can be decomposed into a set of smaller problems by examining the incompatibility graph. Our algorithm will first construct a simple level-$\leq k$ network for each connected component of the incompatibility graph and subsequently merge these simple level-$\leq k$ networks into a single level-k network on all taxa.

Consider a set of taxa $\mathcal{X}$ and a set $\mathcal{C}$ of input clusters. We assume that all singletons (sets $\{x\}$ with $x \in \mathcal{X}$) are clusters in $\mathcal{C}$. Our algorithm proceeds as follows.

Step 1. Find the nontrivial connected components $\mathcal{C}_1, \ldots, \mathcal{C}_p$ of the incompatibility graph $IG(\mathcal{C})$. For each $i \in \{1, \ldots, p\}$, let $\mathcal{C}_i'$ be the result of collapsing unseparated sets of taxa as follows. Let $\mathcal{X}_i = \bigcup_{C \in \mathcal{C}_i} C$. For each maximal subset $X \subset \mathcal{X}_i$ that is not separated by $\mathcal{C}_i$, replace, in each cluster in $\mathcal{C}_i$, the elements of X by a single new taxon X, e.g. if $X = \{b, c\}$ then a cluster $\{a, b, c, d\}$ is modified to $\{a, \{b, c\}, d\}$.

Step 2. For each $i \in \{1, \ldots, p\}$, construct a simple level-$\leq k$ network N_i representing $\mathcal{C}_i'$.

Step 3. Let $\mathcal{C}^*$ be the result of applying the following modifications to $\mathcal{C}$, for each $i \in \{1, \ldots, p\}$: remove all clusters that are in $\mathcal{C}_i$, add a cluster $\mathcal{X}_i$ and add each maximal subset $X \subset \mathcal{X}_i$ that is not separated by $\mathcal{C}_i$. Construct the unique phylogenetic tree T on $\mathcal{X}$ representing precisely those clusters in $\mathcal{C}^*$.

Step 4. For each $i \in \{1, \ldots, p\}$, replace in T the lowest common ancestor v_i of $\mathcal{X}_i$ by the simple level-$\leq k$ network N_i as follows. Delete all edges leaving v_i and merge T with N_i by identifying the root of N_i with v_i and identifying each leaf of N_i labeled X by the lowest common ancestor of the leaves labeled X in T. Output the resulting network.

Notice that Steps 1,3 and 4 are similar to the corresponding steps in algorithms for constructing galled trees (i.e. level-1 networks) and galled networks [8,10,12]. The reason why we use the same set-up in our algorithm, is outlined by Theorem 1. It shows that, when constructing a level-k network displaying a set of clusters, we can restrict our attention to level-k networks that satisfy the *decomposition property* [10], the definition of which we repeat below.

Because a cluster $C \in \mathcal{C}$ can be represented by more than one edge in a network N, an *edge assignment* ϵ is defined as a mapping that chooses for each cluster $C \in \mathcal{C}$ a single tree edge $\epsilon(C)$ of N that represents C. A network N representing $\mathcal{C}$ is said to satisfy the *decomposition property* w.r.t. $\mathcal{C}$ if there exists an edge assignment ϵ such that:

- for any two clusters $C_1, C_2 \in \mathcal{C}$, the edges $\epsilon(C_1)$ and $\epsilon(C_2)$ are contained in the same biconnected component of N if and only if C_1 and C_2 lie in the same connected component of the incompatibility graph $IG(\mathcal{C})$.

Theorem 1. *Let $\mathcal{C}$ be a set of clusters. If there exists a level-k network representing $\mathcal{C}$, then there also exists such a network satisfying the decomposition property w.r.t. $\mathcal{C}$.*

Proof. Let $\mathcal{C}$ be a set of input clusters and N a level-k network representing $\mathcal{C}$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_p$ be the nontrivial connected components of the incompatibility graph $IG(\mathcal{C})$. For each $i \in \{1, \ldots, p\}$, we construct a simple level-$\leq k$ network N_i as follows. Let $\mathcal{X}_i = \bigcup_{C \in \mathcal{C}_i} C$ as before. For each maximal subset $X \subset \mathcal{X}_i$ (with $|X| > 1$) that is not separated by $\mathcal{C}_i$, replace in N an arbitrary leaf labeled by an element of X by a leaf labeled X and remove all other leaves labeled by elements of X. In addition, remove all leaves with labels that are not in X_i. We tidy up the resulting graph by repeatedly applying the following five steps until none is applicable: (1) delete unlabeled nodes with outdegree 0; (2) suppress nodes with indegree and outdegree 1 (i.e. contract one edge incident to the node); (3) replace multiple edges by single edges, (4) remove the root if it has outdegree 1 and (5) contract biconnected components that have only one outgoing edge. This leads to a level-k network N_i. Let $\mathcal{C}_i'$ be defined as in Step 1 of the algorithm. By its construction, N_i represents $\mathcal{C}_i'$. Furthermore, N_i is a simple level-$\leq k$ network, because if it would contain a cut-edge e whose head is not a leaf, then the set of taxa labeling leaves reachable from e would not be separated by $\mathcal{C}_i'$ and would hence

have been collapsed. Finally, the networks $N_1, \ldots, N_p$ can be merged into a level-k network representing $\mathcal{C}$ and satisfying the decomposition property by executing Steps 3 and 4 of the algorithm. $\qquad \square$

Note that the analogous statement obtained by replacing "level-k network" by "network with k reticulations" does not hold, as shown in [12], based on [7].

4 Simple Level-k Networks

This section describes how one can construct a simple level-k network representing a given set of clusters. We say that a phylogenetic tree T is a *strict subtree* of a network N if T is a subgraph of N and for each node v of T, except its root, it holds that the in- and outdegree of v in T are equal to the in- and (respectively) outdegree of v in N.

Informally, our method for constructing simple level-k networks operates as follows. We loop through all taxa x. For each choice for x, we remove it from each cluster and subsequently collapse all maximal "ST-sets" ("strict tree sets", defined below) of the resulting cluster set. We repeat this step k times. The idea behind this strategy is as follows. Observe that any simple level-k network N contains a leaf whose parent is a reticulation. If we would remove this leaf and reticulation from N, the resulting network might contain one or more strict subtrees. Each such strict subtree corresponds to an ST-set. Moreover, for the case $k \leq 2$ we prove that (without loss of generality) each maximal strict subtree corresponds to a maximal ST-set. Collapsing each maximal strict subtree of the network would lead to a (not necessarily simple) level-$(k-1)$ network, which would again contain a leaf whose parent is a reticulation. It follows that we can indeed repeat the described steps k times, after which all leaves will be collapsed into just two taxa and the second phase of the algorithm starts.

We create a network consisting of a root with two children, labeled by the only two taxa. Then we "decollapse", i.e. we replace each leaf labeled by an ST-set by a strict subtree. Subsequently we add a new leaf below a new reticulation and label it by the latest removed taxon. Since we do not know where to create the new reticulation, we try adding the reticulation below each pair of edges. For each constructed simple level-k network, we check whether it represents all input clusters. If it does, we output the resulting network, after contracting any edges that connect two reticulations.

Let us now formalize this algorithm. Given a set $S \subseteq \mathcal{X}$ of taxa, we use $\mathcal{C} \setminus S$ to denote the result of removing all elements of S from each cluster in $\mathcal{C}$ and we use $\mathcal{C}|S$ to denote $\mathcal{C} \setminus (\mathcal{X} \setminus S)$ (the restriction of $\mathcal{C}$ to S). We say that a set $S \neq \mathcal{X}$ is an *ST-set* (strict tree set) w.r.t. $\mathcal{C}$, if S is not separated by $\mathcal{C}$ and any two clusters $C_1, C_2 \in \mathcal{C}|S$ are compatible. An ST-set S is *maximal* if there is no ST-set T with $S \subsetneq T$. Informally, the maximal ST-sets are the result of repeatedly collapsing pairs of unseparated taxa as long as possible.

We use $\textsc{Collapse}(\mathcal{C})$ to denote the result of collapsing each maximal ST-set S into a single taxon S. More precisely, for each cluster $C \in \mathcal{C}$ and maximal ST-set S of $\mathcal{C}$, we replace C by $C \setminus S \cup \{\{S\}\}$. For example (omitting singleton clusters), if

$$\mathcal{C} = \{ \ \{1,2\}, \ \{2,3,4\}, \ \{3,4\} \ \} \ ,$$

then $\{3,4\}$ is the only nonsingleton maximal ST-set and

$$\textsc{Collapse}(\mathcal{C}) = \{ \ \{1,2\}, \ \{2,\{3,4\}\} \ \} \ .$$

The set of taxa of a (collapsed) cluster set $\mathcal{C}$ is denoted $\mathcal{X}(\mathcal{C})$. Thus, for the above example, $\mathcal{X}(\textsc{Collapse}(\mathcal{C})) = \{1, 2, \{3, 4\}\}$. We are now ready to give the pseudocode of $\textsc{Cass}(k)$ in Algorithm 1. The actual implementation is slightly more complex and much more space efficient.

Figure 2 shows how the $\textsc{Cass}(2)$ algorithm for example constructs a simple level-2 network. We will now show that $\textsc{Cass}(1)$ and $\textsc{Cass}(2)$ will indeed construct a simple level-1 respectively level-2 network whenever this is possible.
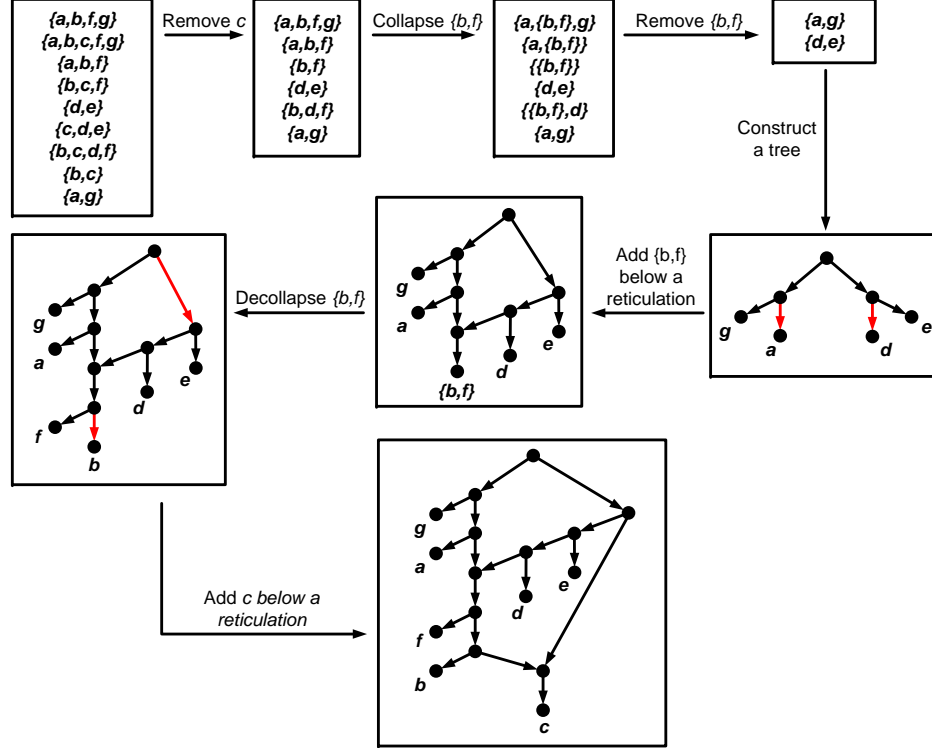
Figure 2. Construction of a simple level-2 network by the CASS algorithm. The edges e_1, e_2 that will be subdivided are colored red. Singleton clusters have been omitted, as well as the last collapse-step, for simplicity.

Lemma 1. *Given a set of clusters* $\mathcal{C}$*, such that* $IG(\mathcal{C})$ *is connected and any* $X \subsetneq \mathcal{X}$ *is separated,* CASS(1) *and* CASS(2) *construct a simple level-1 respectively a simple level-2 network representing* $\mathcal{C}$*, if such a network exists.*

Proof. The general idea of the proof is as follows. Details have been omitted due to space constraints. Assume $k \leq 2$. It is clear that any (simple) level-k network N contains a reticulation r with a leaf, say labeled x, as child. Let $N \setminus \{x\}$ denote the network obtained by removing the reticulation r and the leaf labeled x from N. This network might contain one or more strict subtrees. By the definition of ST-set, the set of leaf-labels of each maximal strict subtree corresponds to an ST-set w.r.t. $\mathcal{C} \setminus \{x\}$. However, in general not each such set needs to be a *maximal* ST-set. This is critical, because the total number of ST-sets can be exponentially large. Therefore, the main ingredient of our proof is the following. We show that whenever there exists a simple level-k network representing $\mathcal{C}$, there exists a simple level-k network N' representing $\mathcal{C}$ such that the sets of leaf-labels of the maximal strict subtrees of $N' \setminus \{x\}$ are the maximal ST-sets w.r.t. $\mathcal{C} \setminus \{x\}$, with x the label of some leaf whose parent is a reticulation in N'. This is clearly true for $k = 1$. For $k = 2$ we sketch our proof below.

Let us first mention that the actual algorithm is slightly more complicated than the pseudocode in Algorithm 1. Firstly, when CASS(k) constructs a tree, it adds a new "dummy" root to this tree and creates an edge from this dummy root to the old root. Such a dummy root is removed before outputting a network. Secondly, whenever the algorithm removes a dummy taxon d, it makes sure that it does not collapse in the previous step.

Suppose there exists some level-2 network representing $\mathcal{C}$. It can be shown that any such network is simple and that there exists at least one binary such network, say N. Since N is a binary simple level-2 network, there are only four possibilities for the structure of N (after removing leaves), see [14]. These structures are called *generators*. In each case, $N \setminus \{x\}$ contains at most two maximal strict subtrees that have more than

---

Algorithm 1 CASS(k): constructing a simple level-k network from clusters

---

1: **input** $(\mathcal{C}, \mathcal{X}, k, k')$
2: **output** CASS$(\mathcal{C}, \mathcal{X}, k, k')$
3: // in the initial call to the algorithm, $k' = k$
4: $\mathcal{N} := \emptyset$
5: **if** $k' = 0$ **then**
6:    **return** the unique tree representing exactly those clusters in $\mathcal{C}$ or return $\emptyset$ if no such tree exists
7: **for** $x \in \mathcal{X} \cup \{d\}$ **do**
8:    // d is a dummy taxon not in $\mathcal{X}$
9:    **remove leaf:** $\mathcal{C}' := \mathcal{C} \setminus \{x\}$
10:    **collapse:** $\mathcal{C}'' := \text{COLLAPSE}(\mathcal{C}')$
11:    **recurse:** $\mathcal{N}' := \text{CASS}(\mathcal{C}'', \mathcal{X}(\mathcal{C}''), k, k' - 1)$
12:    **for** $N' \in \mathcal{N}'$ **do**
13:       **decollapse:** replace each leaf of N' labeled by a maximal ST-set S w.r.t. $\mathcal{C}'$ by the tree on S representing exactly those clusters in $\mathcal{C}'|S$
14:       **for** each pair of edges e_1, e_2 **do**
15:          **add leaf below reticulation:** create a reticulation t, a leaf l labeled x and an edge from t to l;
16:          for $i = 1, 2$, insert a node v_i into e_i and add an edge from v_i to t, this gives network N
17:          **if** N represents $\mathcal{C}$ **then**
18:             **save network:** $\mathcal{N} := \mathcal{N} \cup \{N\}$
19: **if** $k = k'$ **then**
20:    **return** any simple level-k network in $\mathcal{N}$, after removing each leaf labeled d and contracting each edge connecting two reticulations
21: **else**
22:    **return** $\mathcal{N}$

---

one leaf. Furthermore, $N \setminus \{x\}$ contains exactly one reticulation r', below which hangs a strict subtree T_r with set of leaf-labels X_r (possibly, $|X_r| = 1$ or $|X_r| = 0$).

First we assume that X_r is not a maximal ST-set w.r.t. $\mathcal{C} \setminus \{x\}$. In that case it follows that there is some maximal ST-set X that contains X_r and also contains at least one taxon labeling a leaf ℓ that is not reachable by a directed path from the reticulation of $N \setminus \{x\}$. We can replace ℓ by a strict subtree on X that represents $\mathcal{C}|X$. Such a tree exists because X is an ST-set. We remove all leaves that label elements of X and are not in this strict subtree. Since there are now no leaves left below the reticulation, we can remove this reticulation as well. It is easy to see that the resulting network is a tree representing $\mathcal{C} \setminus \{x\}$. Moreover, we show that in each case a leaf labeled x can be added below a new reticulation (possibly with indegree 3) in order to obtain a network N' that represents $\mathcal{C}$. Since N' contains just one reticulation, it is clear that the maximal strict subtrees of $N' \setminus \{x\}$ are the maximal ST-sets w.r.t. $\mathcal{C} \setminus \{x\}$. CASS(2) reconstructs such a network with an indegree-3 reticulation by removing x, removing a dummy taxon d, constructing a tree, adding a leaf labeled d below a reticulation, adding a leaf labeled x below a reticulation, removing the leaf labeled d and contracting the (now redundant) edges between the two reticulations. Note that this works because CASS(2) does not collapse in this case.

It remains to consider the possibility that X_r is a maximal ST-set w.r.t. $\mathcal{C} \setminus \{x\}$. In this case we modify network N to N' in such a way that also the other maximal ST-sets w.r.t. $\mathcal{C} \setminus \{x\}$ appear as the leaf-sets of strict subtrees in $N' \setminus \{x\}$. We again use a case analysis to show that this is always possible in such a way that the resulting network N' represents $\mathcal{C}$.   □

Lemma 2. CASS *runs in time* $O(|\mathcal{X}|^{3k+2} \cdot |\mathcal{C}|)$, *if k is fixed.*

Proof. Omitted due to space constraints.   □

Theorem 2. *Given a set of clusters $\mathcal{C}$, CASS constructs in polynomial time a level-2 network representing $\mathcal{C}$, if such a network exists.*

Proof. Follows from Lemmas 1 and 2 and Theorem 1.   □

Data		GALLEDNETWORK			CASS		
$|\mathcal{C}|$	$|\mathcal{X}|$	t	k	r	t	k	r
30	5	$0s$	6	6	$1s$	4	4
62	6	$0s$	8	8	$7s$	5	5
126	7	$0s$	10	10	$28s$	6	6
254	8	$6s$	12	12	$4m\ 3s$	7	7
42	10	$0s$	4	4	$6s$	4	4
38	11	$0s$	7	7	$14s$	5	5
61	11	$0s$	6	6	$47s$	5	5
77	22	$0s$	9	9	$36s$	3	3
75	30	$0s$	11	11	$5s$	2	2
89	31	$0s$	16	16	$27m\ 32s$	4	4
180	51	$0s$	11	11	$30s$	2	2
193	57	$0s$	1	4	$1s$	1	4
270	76	$0s$	16	16	$4m\ 52s$	2	2
404	122	$1s$	2	2	$21m\ 10s$	2	2
135.8	**31.9**	**1s**	**8.5**	**8.7**	**4m 19s**	**3.7**	**3.9**

Table 1. Results of CASS compared to GALLEDNETWORK for several example cluster sets with $|\mathcal{C}|$ clusters and $|\mathcal{X}|$ taxa. For each algorithm, the level k and reticulation number r of the output network are given as well as the running time t in minutes m and seconds s on a $1.67Ghz$ $2GB$ laptop. The last row gives the average values.

We conclude this section by showing that for each $r \geq 2$, there exists a set of clusters $\mathcal{C}_r$ such that any galled network representing $\mathcal{C}_r$ needs at least r reticulations, while CASS constructs a network with just two reticulations, which also represents $\mathcal{C}_r$. This follows from the following lemma.

Lemma 3. *For each $r \geq 2$, there exists a set $\mathcal{C}_r$ of clusters such that there exists a network with two reticulations that represents $\mathcal{C}_r$ while any galled network representing $\mathcal{C}_r$ contains at least r reticulations.*

Proof. Omitted due to space constraints. □

5 Practice

Our implementation of the CASS algorithm is available as part of the Dendroscope program [9]. To use CASS, first load a set of trees into Dendroscope. Subsequently, run the algorithm by choosing "options" and "network consensus". The program gives you the option of entering a threshold percentage t. Only clusters that appear in more than t percent of the input trees will be used as input for CASS. Choose "minimal network" to run the CASS algorithm to construct a phylogenetic network representing all clusters that appear in more than t percent of the input trees.

CASS computes a solution for each biconnected component separately. If the computations for a certain biconnected component take too long, you can choose to "skip" the component, in which case the program will quickly compute the cluster network [11] for this biconnected component, instead. Alternatively, you can choose to construct a galled network, or to increase the threshold percentage t. For more information on using Dendroscope, see [9].

We have tested CASS on both practical and artificial data and compared CASS to other programs. The results (using $t = 0$) are in Tables 1 and 2. For the former table, several example data sets have been used, which have been selected in such a way as to obtain a good variation in number of taxa, number of clusters and network complexity. For each data set, we have constructed one network using CASS, which we call the CASS-network, and one galled network using the algorithm in [12]. Two conclusions can be drawn from the results. Firstly, CASS uses more time than the galled network algorithm. Nevertheless, the time needed by

Data	$|\mathcal{X}|$	HYBRIDNUMBER		HYBRIDINTERLEAVE		CASS		
		t	r	t	r	t	k	r
ndhF and phyB	40	$11h$	14	$23s$	14	$1s$	4	8
ndhF and rbcL	36	$11h\ 48m$	13	$3s$	13	$0s$	3	8
ndhF and rpoC	34	$26h\ 18m$	12	$6s$	12	$6s$	5	9
ndhF and waxy	19	$5m\ 20s$	9	$1s$	9	$1s$	4	6
ndhF and ITS	46	$> 2d$	?	$4m\ 18s$	19	$> 2d$	?	?
phyB and rbcL	21	$1s$	4	$1s$	4	$0s$	2	4
phyB and rpoC	21	$1m\ 30s$	7	$1s$	7	$0s$	3	4
phyB and waxy	14	$1s$	3	$1s$	3	$0s$	2	3
phyB and ITS	30	$10s$	8	$1s$	8	$1s$	4	8
rbcL and rpoC	26	$15h\ 12m$	13	$8s$	13	$10s$	5	7
rbcL and waxy	12	$2m\ 12s$	7	$1s$	7	$1s$	4	4
rbcL and ITS	29	$> 2d$	?	$10m\ 12s$	14	$> 2d$	?	?
rpoC and waxy	10	$1s$	1	$1s$	1	$0s$	1	1
rpoC and ITS	31	$> 2d$	?	$57s$	15	$> 2d$	?	?
waxy and ITS	15	$10m\ 20s$	8	$1s$	8	$1s$	4	5
Average	**23.2**	**5h 22m**	**8.3**	**4s**	**8.3**	**2s**	**3.5**	**5.7**

Table 2. Results of CASS compared to two HYBRIDNUMBER and HYBRIDINTERLEAVE for several combinations of two input trees with $|\mathcal{X}|$ the number of taxa the two trees have in common, with r the reticulation number, k the level and t the running time in hours h, minutes m and seconds s. The averages do not include the data sets for which HYBRIDNUMBER and CASS did not find a solution within two days (denoted "$> 2d$").

CASS can still be considered acceptable for phylogenetic analysis. Secondly, CASS constructs a much simpler network in almost all cases. For three data sets, the CASS-network and the galled network have the same reticulation number and the same level. For all other data sets, the CASS-network has a significantly smaller reticulation number, and also a lower level, than the galled network.

In Table 2 are the results of an application of CASS to practical data. This data set consists of six phylogenetic trees of grasses of the *Poaceae* family, originally published by the Grass Phylogeny Working Group [6]. The phylogenetic trees are based on sequences from six different gene loci, ITS, ndhF, phyB, rbcL, rpoC and waxy, and contain 47, 65, 40, 37, 34 and 19 taxa respectively. We have compared the results of CASS with results of HYBRIDNUMBER [1], which is a program that computes the minimum number of reticulations required to combine two phylogenetic trees (on the same set of taxa) into a phylogenetic network that displays both trees. Since HYBRIDNUMBER can only be used for pairs of trees with identical taxon sets, we have also applied CASS to pairs of trees and restricted each of the two trees to the taxa that appear in both. Table 2 shows that the running time of CASS is much better than the running time of HYBRIDNUMBER. Very recently, an improved version of HYBRIDNUMBER, called HYBRIDINTERLEAVE [4], was published, which is significantly faster than the original program.

Table 2 shows that CASS requires significantly fewer reticulations than HYBRIDNUMBER and HYBRID-INTERLEAVE. This is caused by the fact that the latter programs require that a network displays both input trees. The networks constructed by CASS do not necessarily display both input trees, but still represent all clusters from both trees, and use fewer reticulations to do so. Other advantages of CASS are that it can also be used for cluster sets that are obtained from more than two trees and for cluster sets obtained from trees on nonidentical sets of taxa. Moreover, while HYBRIDNUMBER and HYBRIDINTERLEAVE only compute the required number of reticulations, CASS also constructs an actual network. See for example Figure 3 for the output network of CASS for the ndhF and phyB trees of the *Poaceae* data set.

6   Discussion

We have introduced the CASS algorithm, which can be used to combine any set of clusters into a phylogenetic network representing those clusters. We have shown that the algorithm performs well on practical
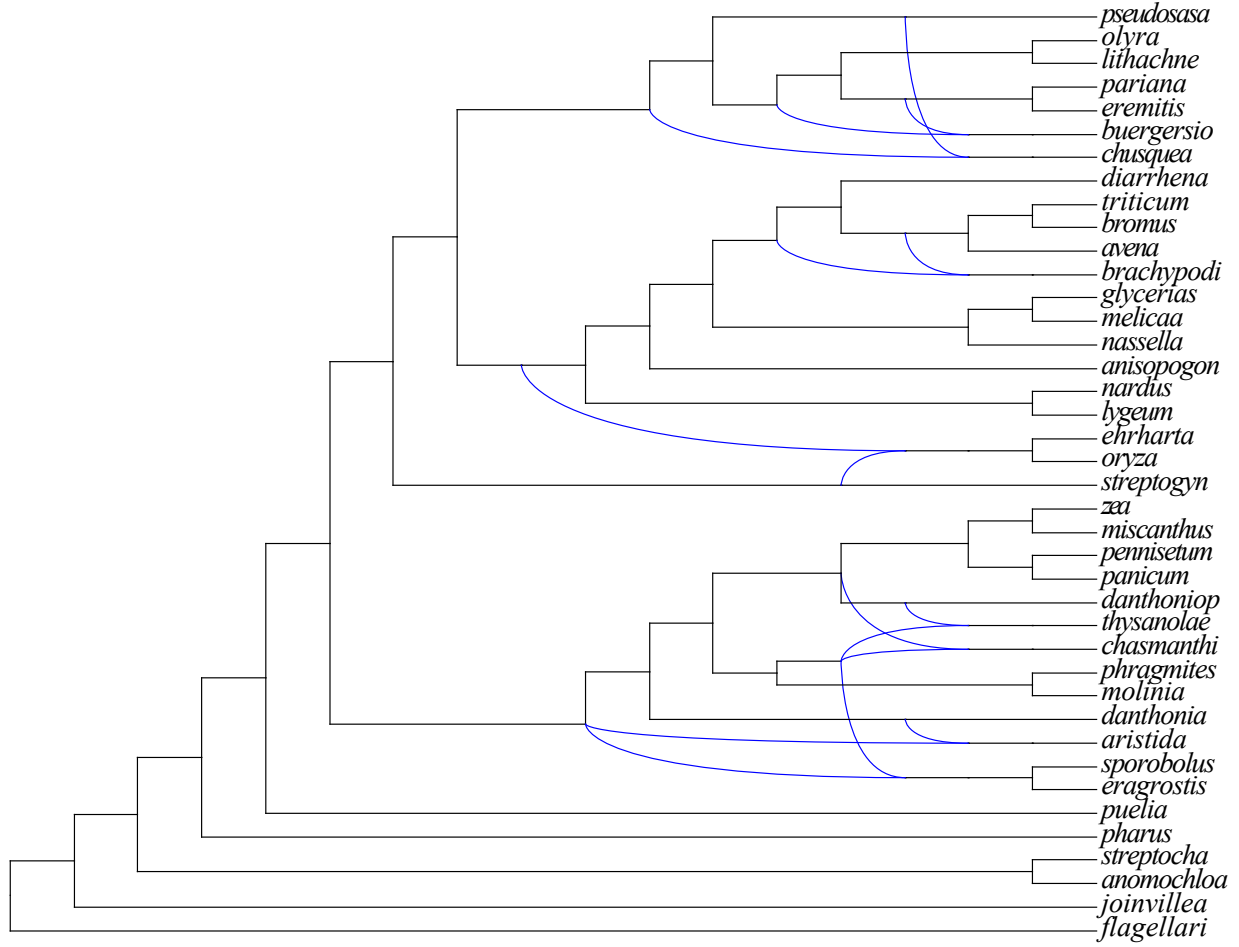
Figure 3. Level-4 network constructed by CASS for the ndhF and phyB trees of the *Poaceae* grass data set, within 1 second.

data. It provides a useful addition to existing software, because it usually constructs a simpler network representing the same set of input clusters. Furthermore, we have shown that CASS provides a polynomial-time algorithm for deciding whether a level-2 phylogenetic network exists that represents a given set of clusters. This algorithm is more useful in practice than algorithms for similar problems that take triplets as input [14,15,16,17,19], because the latter algorithms need at least one triplet for each combination of three taxa as input, while CASS can be used for any set of input clusters. Furthermore, CASS is also not restricted to two input trees on identical taxon sets, as the algorithms in [1,4,18]. Finally, we remark that CASS can also be used when one or more multi-labeled trees are given as input. One can first compute all clusters in the multi-labeled tree(s) and subsequently use CASS to find a phylogenetic network representing these clusters. Several theoretical problems remain open. First of all, does CASS always construct a minimum-level network, even if this minimum is three or more? Secondly, what is the complexity of constructing a minimum level network, if the minimum level k is not fixed but part of the input? Is this problem FPT when parameterized by k? Finally, it would be very interesting to design an algorithm that finds a network representing a set of input clusters that has a minimum reticulation number. So far, not even a nontrivial exponential-time algorithm is known for this problem.

Acknowledgements

References

1. M. Bordewich, S. Linz, K. S. John, and C. Semple. A reduction algorithm for computing the hybridization number of two trees. *Evolutionary Bioinformatics*, 3:86–98, 2007.
2. M. Bordewich and C. Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.
3. C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung. Computing the maximum agreement of phylogenetic networks. *Theoretical Computer Science*, 335(1):93–107, 2005.
4. J. Collins, S. Linz, and C. Semple. Quantifying hybridization in realistic time. To appear.
5. P. Gambette. Who's who in phylogenetic networks, 2009. http://www.lirmm.fr/~gambette/PhylogeneticNetworks/.
6. Grass Phylogeny Working Group. Phylogeny and subfamilial classification of the grasses (*Poaceae*). *Annals of The Missouri Botanical Garden*, 88:373–457, 2001.
7. D. Gusfield, V. Bansal, V. Bafna, and Y. S. Song. A Decomposition Theory for Phylogenetic Networks and Incompatible Characters. *Journal of Computational Biology*, 14(10): 1247-1272, 2007.
8. D. H. Huson and T. H. Klöpper. Beyond galled trees - decomposition and computation of galled networks. In *Research in Computational Molecular Biology (RECOMB)*, volume 4453 of *Lecture Notes in Computer Science*, pages 221–225, 2007.
9. D. H. Huson, D. C. Richter, C. Rausch, T. Dezulian, M. Franz, and R. Rupp. Dendroscope: An interactive viewer for large phylogenetic trees. *BMC Bioinformatics*, 8:460, 2007. http://www.dendroscope.org
10. D. H. Huson and R. Rupp. *Phylogenetic Networks*. Cambridge University Press. To appear.
11. D. H. Huson and R. Rupp. Summarizing multiple gene trees using cluster networks. In *Algorithms in Bioinformatics (WABI)*, volume 5251 of *Lecture Notes in Bioinformatics*, pages 296–305, 2008.
12. D. H. Huson, R. Rupp, V. Berry, P. Gambette, and C. Paul. Computing galled networks from real data. *Bioinformatics*, 25(12):i85–i93, 2009.
13. T. Huynh, J. Jansson, N. Nguyen, and W.-K. Sung. Constructing a smallest refining galled phylogenetic network. In *Research in Computational Molecular Biology (RECOMB)*, volume 3500 of *Lecture Notes in Bioinformatics*, pages 265–280, 2005.
14. L. J. J. van Iersel, J. C. M. Keijsper, S. M. Kelk, L. Stougie, F. Hagen, and T. Boekhout. Constructing level-2 phylogenetic networks from triplets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2009. To appear.
15. L. J. J. van Iersel and S. M. Kelk. Constructing the simplest possible phylogenetic network from triplets. *Algorithmica*, 2009. To appear.
16. J. Jansson, N. B. Nguyen, and W.-K. Sung. Algorithms for combining rooted triplets into a galled phylogenetic network. *SIAM Journal on Computing*, 35(5):1098–1121, 2006.
17. J. Jansson and W.-K. Sung. Inferring a level-1 phylogenetic network from a dense set of rooted triplets. *Theoretical Computer Science*, 363(1):60–68, 2006.
18. S. Linz and C. Semple. Hybridisation in nonbinary trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6:30–45, 2009.
19. T.-H. To and M. Habib. Level-k phylogenetic networks are constructable from a dense triplet set in polynomial time. In *Combinatorial Pattern Matching (CPM)*, volume 5577 of *Lecture Notes in Computer Science*, pages 275–288, 2009.

A Appendix

Lemma 1. Given a set of clusters $\mathcal{C}$, such that $IG(\mathcal{C})$ is connected and any $X \subsetneq \mathcal{X}$ is separated, CASS(1) and CASS(2) construct a simple level-1 respectively a simple level-2 network representing $\mathcal{C}$, if such a network exists.

Proof. We start by proving the following claim. We assume that networks do not contain biconnected components with only one outgoing edge (because such structures are highly redundant). Furthermore, in this proof we identify each leaf with the taxon it is labeled by, to shorten the notation.

Claim. Given a set of clusters $\mathcal{C}$, such that $IG(\mathcal{C})$ is connected and any $X \subsetneq \mathcal{X}$ is separated, then any network N representing $\mathcal{C}$ is simple and no two leaves in N have the same parent. Additionally, if such a network N exists, then there also exists a *binary* simple network N' representing $\mathcal{C}$ which has the same level as N and such that no two leaves of N' have the same parent.

Proof. If N is not simple then it contains a cut-edge (v_1, v_2) such that v_2 is not a leaf and some subset of the taxa $\mathcal{X}' \subsetneq \mathcal{X}$ is reachable by directed paths starting at v_2. Since we assume that networks do not contain biconnected components with only one outgoing edge, $|\mathcal{X}'| \geq 2$. Now, given that $\mathcal{X}'$ is below a cut-edge, it follows that for every cluster $C \in \mathcal{C}$ holds that either $\mathcal{X}' \subseteq C$ or $C \subseteq \mathcal{X}'$. So $\mathcal{X}'$ is unseparated, giving us an immediate contradiction. To prove the second half of the lemma we show how to obtain N' from N by expanding edges. First we deal with nodes v that have both indegree and outdegree greater than 1. Here we replace the node v by an edge (v_1, v_2) such that the edges incoming to v now enter v_1, and the edges outgoing from v now exit from v_2. Subsequently nodes with indegree at most 1, and outdegree $d \geq 3$, can be replaced by a chain of $(d-1)$ nodes of indegree at most 1 and outdegree 2. Nodes with indegree $d \geq 3$ and outdegree 1 can be replaced by a chain of $(d-1)$ nodes of indegree 2 and outdegree 1. These transformations preserve the reticulation number of the network and do not introduce any nontrivial cut-edges (i.e. cut-edges that do not have a leaf as head), so the resulting network N is a binary simple network with the same level as N'. Binary simple networks cannot contain sibling leaves, so we are done. $\square$

Before continuing we need some definitions. We say that a tree-edge $e = (v_1, v_2)$ (i.e. an edge where v_2 is not a reticulation) of a network N is *contraction-safe* for $\mathcal{C}$ if N represents $\mathcal{C}$ and there is no $C \in \mathcal{C}$ that is represented by e. (An edge (u, v) of N is said to represent a cluster C if there exists a tree T on $\mathcal{X}$ that is displayed by N, and such that C consists of all taxa reachable by a directed path from v in T.) Clearly such an edge can be contracted to obtain a new network N' that still represents $\mathcal{C}$.

We are now ready to prove the lemma. Suppose we are given a set of clusters $\mathcal{C}$, such that $IG(\mathcal{C})$ is connected and any $X \subsetneq \mathcal{X}$ is separated.

It is clear that CASS(0) will return in polynomial-time a tree T representing $\mathcal{C}$, if it exists. In this case T will be the unique tree that represents $\mathcal{C}$ and which contains no contraction-safe edges.

We now show that CASS(1) will return in polynomial time a simple level-1 network that represents $\mathcal{C}$, if it exists. Suppose then that such a network, N, exists. We assume that N is a binary simple level-1 network. CASS will thus at some iteration correctly guess and remove the (unique) leaf x whose parent is a reticulation in N. CASS will then construct the unique (and in general non-binary) tree T that represents $\mathcal{C} \setminus \{x\}$ and which contains no contraction-safe edges. To complete the level-1 case it is necessary to show that x can be hung back from two edges of T (in the sense of lines 14-16 of the CASS pseudocode) to create a level-1 network representing $\mathcal{C}$. In [14] it is described how, after removal of leaves, a binary simple level-k network always has a topology equal to one of the binary *simple level-k generators*, depicted in Figure 4. We repeat the following definition [14].

Definition 1. *[14] A simple level-k network N, for $k \geq 1$, is a network obtained by applying the following transformation ("leaf hanging") to some simple level-k generator such that the resulting graph is a valid network:*
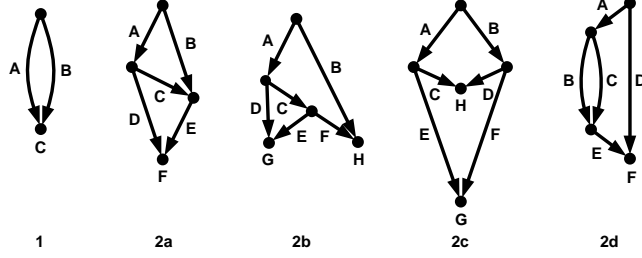
Figure 4. The single level-1 generator and the four level-2 generators.

1. *replace each edge X by a path and for each internal node v of the path add a new leaf x and an edge (v, x); we say that "leaf x is on side X"; and*
2. *for each node Y of indegree 2 and outdegree 0 add a new leaf y and an edge (Y, y); we say that "leaf y is on side Y".*

Consider in particular that N is constructed from the unique level-1 generator 1. There are two cases. In the first case N has leaves on both sides A and B, in which case let a (respectively b) be the leaf on side A (respectively B) that is furthest from the root. We hang x from the edges in T that feed into a and b respectively, to obtain the network N'. We refer to the two corresponding reticulation edges as the a- (respectively, b-) *reticulation edge*. Consider a cluster $C \in \mathcal{C}$. If $\{x, a\} \subseteq C$ or $\{x, a\} \cap C = \emptyset$ then we see that N' represents C because T represented $C \setminus \{x\}$ and we can simply "switch on" the a-reticulation edge. If $\{x, a\} \cap C = \{a\}$ then $b \notin C$, and $b \notin C \setminus \{x\}$, so we can switch on the b-reticulation edge. If $\{x, a\} \cap C = \{x\}$ then C was either a singleton (which is trivially represented) or $b \in C$, in which case we can switch on the b-reticulation edge. In the second and final case, assume without loss of generality that only side A contains leaves. Let a be defined as before. We hang x back from the edge feeding into a in T, and from a new root node that we also connect to the old root of T. Clusters in C of the form $\{x, a\} \subseteq C$ or $\{x, a\} \cap C = \emptyset$ are dealt with as before. If $\{x, a\} \cap C = \{x\}$ then C is a singleton. If $\{x, a\} \cap C = \{a\}$ then we can switch the reticulation edge leaving the new root on and we are done.

There remains only the case that there exists a simple level-2 network representing $\mathcal{C}$, but no tree or simple level-1 network. This case is rather complex and requires some extra terminology, although the central idea has much in common with the proof for simple level-1 networks that we have just presented. Given a network N and reticulation r with a leaf x as child, let $N \setminus \{x\}$ denote the network obtained by removing the reticulation r and the leaf x from N. We say that a network N is *drooped with respect to $\mathcal{C}$* if N represents $\mathcal{C}$ and there exists a leaf x of N whose parent is a reticulation, such that the leaf-sets of maximal strict subtrees of $N \setminus \{x\}$ correspond to the maximal ST-sets of $\mathcal{C} \setminus \{x\}$.

We are going to first prove that, if there is a simple level-2 network N representing $\mathcal{C}$, then there exists a simple level-2 network N' that is drooped w.r.t. $\mathcal{C}$. We do this partially by case analysis on the four possible generator topologies for N shown in Figure 4. The general strategy will be to argue that either N is already drooped, or that it can be transformed into some new level-2 network N' representing $\mathcal{C}$. If N' is subsequently level-1 and/or not simple then we obtain a contradiction. Otherwise, N' is (as we will demonstrate) a drooped simple level-2 network.

Consider any leaf x whose parent is a reticulation of N. Observe that $N \setminus \{x\}$ contains exactly one reticulation r', below which hangs a strict subtree T_r with leaves X_r (possibly, $|X_r| = 1$ or $|X_r| = 0$). Note that, if X_r is the empty set then N contains an edge between two reticulations and contracting this edge leads to a network N' that is drooped w.r.t. $\mathcal{C}$, because $N' \setminus \{x\}$ is a tree. We distinguish two major cases.

First major case: X_r is not a maximal ST-set w.r.t. $\mathcal{C} \setminus \{x\}$ and not the empty set.

In this case it follows that there is some maximal ST-set X that contains X_r and also contains at least one leaf ℓ that is not reachable by a directed path from the reticulation of $N \setminus \{x\}$. We can replace ℓ by a

strict subtree on X that represents $\mathcal{C}|X$. Such a tree exists because X is an ST-set. We remove all leaves in X that are not in this strict subtree. Since there are now no leaves left below the reticulation, we can remove this reticulation as well. Let N^* be the resulting network. It is easy to see that N^* is a tree representing $\mathcal{C} \setminus \{x\}$. We now require a case-analysis to show how the leaf x can be hung back into N^* to obtain a network N' that is drooped w.r.t. $\mathcal{C}$.

Case generator 2a

Here we assume that the leaf x is equal to the leaf on side F of N.

Let p be (in N) the leaf on side E that is furthest from the root. Such a leaf will definitely exist by assumption that X_r is not empty. The common core of the construction, independent of the exact case, requires locating p in N^* and hanging x back from the edge that feeds into p. We call this reticulation edge the p-edge. Depending on the exact case we will additionally hang back x from one or two other places (i.e. to create an indegree-2 or indegree-3 reticulation respectively). But note already that, however we add these additional reticulation edges, a cluster $C \in \mathcal{C}$ such that $\{p, x\} \subseteq C$ or $\{p, x\} \cap C = \emptyset$ will definitely be represented by N'. The argument for this is identical to that used in the level-1 case. So we only need to worry about non-singleton clusters C where $\{p, x\} \cap C = \{p\}$ or $\{p, x\} \cap C = \{x\}$. We hang a second reticulation edge of x from the root of N^* and call this the root-edge. Now, let l be the leaf on side D (of N) that is furthest from the root of N; if such a leaf does not exist then let l be the leaf on side C (of N) that is furthest from the root of N, and if that also does not exist then let l be the leaf on side A (of N) that is furthest from the root of N. For brevity we will henceforth abbreviate this specification of l to "the lowest leaf on sides $D; C; A$". If l exists (in general it might not) then hang a third reticulation edge of x from the edge in N^* that feeds into l, call this the l-edge. Consider then a non-singleton cluster C in $\mathcal{C}$ that contains x but not p. Then l exists and cluster C definitely contained it, so $C \setminus \{x\}$ contained l and was represented by N^*. So in N' (the network we get after adding x below a reticulation) we can switch on the l-edge to obtain cluster C. Finally, consider a cluster C that contained p, but not x. Then switching on the root-edge is sufficient. So N' represents $\mathcal{C}$.

If N' is a level-1 network then we have a contradiction. Otherwise it is a drooped simple level-2 network (because by the earlier claim all networks that represent C are simple).

Case generator 2b

Here we assume that the leaf x is the leaf on side G of N. Let l be the lowest leaf on sides $D; A$ of N. Let p be the lowest leaf on sides $E; F; C; A$. We hang x back below a new reticulation of indegree 2 or 3. More specifically: we hang x from the edges that feed into l, p and h: the leaf on side H. If l and p both exist and $l \neq p$ then this reticulation clearly has indegree-3. If $l = p$ or at least one of l and p does not exist then we also hang p back from the root. (The only situation when the reticulation has indegree-2 is thus if neither l nor p exists). Consider now clusters in $\mathcal{C}$. We distinguish several cases: in the case that neither l nor p existed we get a level-1 network (and thus a contradiction), otherwise a drooped level-2 network.

Suppose that the leaf l existed. Then clusters C of the form $\{x, l\} \subseteq C$ or $\{x, l\} \cap C = \emptyset$ are (as in earlier cases) easy to deal with. For a non-singleton cluster C such that $\{x, l\} \cap C = \{x\}$ holds that $p \in C$ and/or $h \in C$. Then in N' we can switch on the p-edge or the h-edge depending on which one is relevant. For a non-singleton cluster C such that $\{l, x\} \cap C = \{l\}$ holds that $\{h, p\} \cap C = \emptyset$, so in N' we can switch the h-edge on.

Suppose, alternatively, that the leaf p existed. Again, "both p and x" and "neither p nor x" clusters are easy to deal with. So, consider a cluster that contains x but not p. Then this cluster will contain either l or h and we are done. Consider a cluster that contains p but not x. If l exists then it will not be in the cluster, so we are done. If l does not exist then we can use the root-edge in N' and we are done.

The final case is that neither l nor p exists. Clusters that contain h and x, or neither h nor x, are easy to deal with. So, consider a non-singleton cluster that contains x but not h. But the sides D, A, E, F, C are all empty, meaning that the cluster is a singleton, contradiction. For clusters that contain h but do not contain

x we can use the root-edge in N', and we are done.

Case generator 2c

Here we assume that the leaf x is the leaf on side G. We assume without loss of generality that N contains at least three leaves. (If N contains only two leaves then N is clearly already drooped). Let l be the lowest leaf (in N) on $E; C; A$. Let p be the lowest leaf (in N) on $F; D; B$. Note that (by the assumption that there are at least three leaves in N) at least one of l and p will exist. If they both exist then hang x back from the edges feeding into l, p and h: the leaf on side H. If (without loss of generality) only l exists then hang x back from the edges feeding into l, h and also the root. Consider then non-singleton clusters in $\mathcal{C}$ that contain x but not l. Then the cluster either contains h or p, and we are done. Finally consider a cluster that contains l but not x. If p exists then it will not be in the cluster, so we are done. If p does not exist then we can use the root-edge.

Case generator 2d

As in case 2a we assume that there is at least one leaf on side E (because otherwise N was already drooped). We assume that the leaf x we removed is the leaf on side F of N. Let l be the lowest leaf on side E in N (which must exist). Let p be the lowest leaf on side D in N. If p exists then we can hang back x from the edges feeding into l and p. Otherwise we hang x back from the edge feeding into l, and the root. Consider then non-singleton clusters in $\mathcal{C}$ that contain x but not l. Then p must exist, and the cluster must contain it. For clusters that contain l but not x we can either use the p-reticulation edge (because it is not possible to contain l and p but not x) or use the root reticulation edge in N'. N' is thus level-1, and we have a contradiction.

This concludes the case analysis for the generators 2a, 2b, 2c and 2d for the first major case i.e. when X_r is not a maximal ST-set or the empty set.

Second major case: X_r is a maximal ST-set w.r.t. $\mathcal{C} \setminus \{x\}$.

Here we argue that either N is already drooped w.r.t $\mathcal{C}$ (i.e. for some reticulation leaf x not only X_r, but also all other maximal ST-sets of $\mathcal{C} \setminus \{x\}$, correspond to strict subtrees of $N \setminus \{x\}$) or that it is possible to transform N into a network N' with this property. We again use a generator-based case analysis.

To shorten the proofs we introduce abbreviations for several commonly-used concepts. If we say *"hang x back from $l_1, \ldots, l_i$"* $(i \geq 2)$ where $l_1, \ldots, l_i \in \mathcal{X}$ we mean: (1) introduce x into the network as the only child of a new reticulation r_x, (2) for each l_j subdivide the unique edge feeding into l_j to create a new node v_j, and finally (3) for each l_j we add the reticulation edge (v_j, r_x). If we say *"hang x back from $l_1, \ldots, l_i$ and the root"* $(i \geq 1)$ this is defined identically except that after steps (1)-(3) we additionally add an edge with head r_x and tail the root. As in earlier proofs we will make heavy use of the fact that, if x is hung back from (amongst others) l_j to obtain a network N', then clusters $C \in \mathcal{C}$ for which $C \cap \{l_j, x\} = \{l_j, x\}$ or $C \cap \{l_j, x\} = \emptyset$, are easily seen to be represented by N'. That is because in such cases the reticulation edge (v_j, x) can simply be "switched on". Clusters C of the form $C \cap \{l_j, x\} = \{l_j\}$ or $C \cap \{l_j, x\} = \{x\}$ require a little more work and in each case will be verifiable by inspection.

If we *"tidy up"* a network we repeatedly apply the following five steps until none is applicable: (1) delete unlabeled nodes with outdegree 0; (2) suppress nodes with indegree and outdegree 1 (i.e. contract one edge incident to the node); (3) replace multiple edges by single edges, (4) remove the root if it has outdegree 1 and (5) contract biconnected components that have only one outgoing edge. Note that tidying up does not affect the set of clusters that a network represents and is simply a housekeeping measure.

If we say, *"move maximal ST-sets below cut-edges"* we refer to the following (fundamental) procedure. Suppose $N \setminus \{x\}$ represents $\mathcal{C} \setminus \{x\}$. Suppose there is a non-singleton maximal subset S of $\mathcal{C} \setminus \{x\}$ that does not correspond to a strict subtree of $N \setminus \{x\}$. Then we can pick any leaf l of S, delete all leaves $l' \in S \setminus \{l\}$ from $N \setminus \{x\}$, replace l with the unique tree that represents precisely those clusters in $\mathcal{C}|S$, and tidy up.

This creates a new network in which S *does* appear as a strict subtree and (because S is an ST-set) still represents $\mathcal{C} \setminus \{x\}$. We can repeat this process until all such S appear as strict subtrees of the final network. In other words, until every maximal ST-set is equal to the set of leaves reachable from some cut-edge. Note that, crucially, this procedure will not affect singleton maximal ST-sets or (in this case) X_r because these already correspond to strict subtrees of the network.

We say *"remove x and transform"* to refer to the combined process of removing x and its parent (from N) to obtain $N \setminus \{x\}$, tidying this network up and subsequently moving all maximal ST-sets (of $\mathcal{C} \setminus \{x\}$) below cut-edges.

Case generator 2a

Let f be the leaf on side F. Let a, b, d, e be the leaf on respectively side A, B, D, E that is furthest from the root. Let c be the leaf on side C that is *closest* to the root. Leaf e must exist, because otherwise N was already drooped. We take $x = f$. We distinguish two subcases. In one subcase we construct a drooped network N' by removing x and transforming, and then hanging x back in such a way that a network is created that represents all clusters in $\mathcal{C}$. This network will be drooped (because, prior to hanging x back, we moved the maximal ST-sets of $\mathcal{C} \setminus \{x\}$ under cut-edges) and thus we are done. In the second case we will show that N was already drooped. For the first case, suppose that leaf d exists. It is easy to see that (after removing x and transforming) hanging x back from e and d creates a drooped network w.r.t. $\mathcal{C}$; the argumentation (e.g. regarding the four possibilities for $C \cap \{e, x\}$ for each $C \in \mathcal{C}$) is identical to that used in the previous proofs, and we are done. If d does not exist, and c does not exist, but a does exist, then we hang x back from e and a, done. If none of d, c, a exist we hang x back from e and the root, done. This leaves us with the case that d does not exist but c does exist. We observe that hanging x back from e and c creates a drooped network N' that is consistent with all clusters except with the possible exception of a cluster C that (in N) contains c and all leaves on side E, but not f or any leaves from side A. If such a cluster does not exist then we are done.

Assume, then, that it does exist. Observe that if we had hung x back from e, c *and* the root then we would have created a (potentially level-3) drooped network w.r.t. $\mathcal{C}$. Let N^* be the network obtained by tidying up $N \setminus \{x\}$. We observe that $\mathcal{C} \setminus \{x\}$ contains at most one non-singleton maximal ST-set that is not equal to X_r. Suppose the opposite was true i.e. that $\mathcal{C} \setminus \{x\}$ contained at least two non-singleton maximal ST-sets not equal to X_r. If we then moved maximal ST-sets under cut-edges we would create a network with at least two nontrivial cut-edges (excluding the cut-edge associated with the strict subtree corresponding to X_r). Hanging x back from e, c and the root in this network would create a network N' that represents $\mathcal{C}$ but which contains at least one nontrivial cut-edge. But the set of leaves reachable by a directed path from such a nontrivial cut-edge forms an unseparated subset of $\mathcal{X}$, which by assumption is not possible. Now, if $\mathcal{C} \setminus \{x\}$ contains no non-singleton maximal ST-sets not equal to X_r then we are immediately done, because N was already drooped. So we conclude that there is exactly one non-singleton maximal ST-set S of $\mathcal{C} \setminus \{x\}$ that is not equal to X_r, and that this must contain c. Note that, because of the existence of cluster C (in particular the fact that cluster C contains leaves from side E, and that X_r is already a maximal ST-set), S must be entirely contained within the leaves of side C (in N). S must thus contain at least two leaves c_1, c_2 on side C. Let c_1 and c_2 be the leaves in S that are furthest from the root, with c_2 furthest away. Some cluster C' separated c_1 from c_2 in $\mathcal{C}$ and this proves that c_1 and c_2 are the last two leaves on side C. (Otherwise $C' \setminus \{x\}$ would prevent S from being a maximal ST-set of $\mathcal{C} \setminus \{x\}$). We conclude (again, by the separation of c_1 and c_2) that C' contained leaves from side E. But then $C' \setminus \{x\}$ prevents S from being a maximal ST-set of $\mathcal{C} \setminus \{x\}$. We conclude thus that $\mathcal{C} \setminus \{x\}$ actually contains no non-singleton maximal ST-sets, with the possible exception of X_r. So N was already drooped.

Case generator 2b

Let g, h be the leaves on sides G, H respectively. Let a be the leaf on side A furthest from the root, and define $b, \ldots, f$ similarly. If f exists then take $x = h$. We remove x and transform and then hang x back from f and b (if b exists) or otherwise from f and the root. A simple case-analysis shows that the resulting network is drooped. So assume that side F has no leaves. If leaf e exists then take $x = g$, remove x and

transform as above, and hang x back from e and d if they both exist, otherwise e and the root. This again gives a network that is drooped w.r.t. $\mathcal{C}$. So assume side E also contains no leaves. Suppose that side C contains no leaves. Then we take $x = h$ and (after removing x and transforming) hang back from b and g (if b exists) and otherwise from g and the root. So there is at least one leaf on side C. If c is the only leaf on side C and none of the clusters $\{c, g\}, \{c, h\}, \{c, g, h\}$ are in $\mathcal{C}$, then we can safely move c to the top of side D, and we are back in the case when side C contains no leaves, done. If side C contains more than one leaf then at least one of $\{c, g\}, \{c, h\}, \{c, g, h\}$ has to be in $\mathcal{C}$, because otherwise c is unseparated from the leaf immediately above it. If $\{c, g\} \in \mathcal{C}$ then take $x = h$, otherwise take $x = g$. First suppose $x = h$. Observe that $\{c\}$ is a maximal ST-set in $\mathcal{C} \setminus \{x\}$ because $\{c, g\} \in \mathcal{C} \setminus \{x\}$ and (by assumption) $\{g\}$ is a maximal ST-set in $\mathcal{C} \setminus \{x\}$. Thus, when we remove x and transform, then neither c nor g will move in the sense of moving maximal ST-sets under cut-edges. This is a critical fact. So, after the removal of x and transformation the path of length two in N between the parent c' of c and the parent g' of g will have been suppressed (by the tidying up) to become a single edge (c', g'). We will now further expand the current notion of "hanging x back", which already defines "hanging back" from leaves and the root, to also define hanging back from an edge (u, v). When we hang x back from the edge (u, v) we subdivide (u, v) to create the new node u' and add the reticulation edge (u', r_x) (where r_x is defined as earlier). We will hang x back from b and (c', g') (if b exists) and otherwise from (c', g') and the root. By inspection it can be verified that this gives a drooped network that represents $\mathcal{C}$. Symmetrically, if $x = g$ then we hang x back from d (if it exists, otherwise the root) and (c', h') where c' is the parent of c and h' is the parent of h (because, again, $\{c\}$ and $\{h\}$ are maximal ST-sets that do not move in the sense of moving maximal ST-sets under cut-edges). Again we obtain a drooped network w.r.t. $\mathcal{C}$, and we are done. Note that the added complexity of this proof comes from the possible presence of clusters $\{g, h\}$ in the input: this is why we have to hang one of the two reticulation edges of x from a carefully identified edge, rather than (as usual) a leaf or the root.

Case generator 2c

Assume $a, \ldots, g, h$ are defined as in case 2b. Let l be the lowest leaf (in N) on $E; C; A$. Let p be the lowest leaf (in N) on $F; D; B$. At least one of l and p will exist, because we assume that N has at least three leaves. (Otherwise it is already drooped). Suppose $\{g, h\} \notin \mathcal{C}$. Then we can take $x = g$, remove x and transform, and then hang x back from l and p (if they both exist), or from l and the root (if l exists), or from p and the root (if p exists). This will give a drooped network w.r.t. $\mathcal{C}$. So assume $\{g, h\} \in \mathcal{C}$. We assume then, without loss of generality, that sides D and F contain no leaves. Suppose that side B also contains no leaves. In this case we take $x = g$, remove x and transform, then hang x back from l (which must exist) and the edge $(root, h')$ where h' is the parent of h. (Again, this edge was originally a path of length 2 in N that was subsequently suppressed by the tidying-up operation). This edge is definitely present because $\{h\}$ is (by assumption) a maximal ST-set of $\mathcal{C} \setminus \{x\}$ and thus remains unaffected by the moving of maximal ST-sets under cut-edges. This creates a drooped network. So we assume that side B contains at least one leaf i.e. that leaf b exists. If b is the only leaf on side B and none of the clusters $\{b, g\}, \{b, h\}, \{b, g, h\}$ are in $\mathcal{C}$ then we could move b to the top of side A and we are back in the case that side B contains no leaves, done. If side B contains more than one leaf then at least one of those clusters must be present, otherwise b and the leaf immediately above it were unseparated. If $\{b, h\} \in \mathcal{C}$ take $x = g$, otherwise take $x = h$. First suppose $x = g$. As in case 2b we argue that $\{b\}$ and $\{h\}$ are maximal ST-sets in $\mathcal{C} \setminus \{x\}$ and thus that the edge (i.e. suppressed path) connecting the parent b' of b to the parent h' of h is unaffected by the movement of maximal ST-sets. In this case we hang x back from (b', h') and from l (if it exists: otherwise the root). It is not too difficult to verify that the resulting network is drooped w.r.t. $\mathcal{C}$. The case $x = h$ is almost entirely symmetrical except that we must redefine l to be the lowest leaf on $C; E; A$ (instead of $E; C; A$).

Case generator 2d

Take $x = f$, where f is the leaf on side F. Let e be the leaf on side E that is furthest from the root. (Leaf e must exist because otherwise $X_r = \emptyset$). Let d be the leaf on side D that is furthest from the root. We remove x and transform, and hang x back from e and d (if d exists) and otherwise from e and the root. This creates a drooped network.

This concludes the second major case, and we have thus proven that a drooped simple level-2 network N' exists that represents $\mathcal{C}$. To complete the overall proof we need to show that CASS will (re)construct N', or some other simple level-2 network representing $\mathcal{C}$. Suppose we contract all contraction-safe edges of N' to obtain N''. In some iteration, CASS correctly identifies a leaf x whose parent is a reticulation r and the maximal ST-set X_r which is the set of leaves below the only reticulation r' of $N'' \setminus \{x\}$.

Let T'' be the tree obtained by removing x, r, X_r and r' from N'' and contracting edges entering unlabeled tree-nodes with outdegree at most 1. We first show that T'' is identical to the unique tree T that represents $\mathcal{C} \setminus (X_r \cup \{x\})$ and which contains no contraction-safe edges. (T is the tree that CASS constructs in its innermost iteration). Suppose $T'' \neq T$. Then T'' will contain at least one contraction-safe edge, implying that N'' also contains at least one contraction-safe edge, yielding a contradiction.

If $X_r \neq \emptyset$, one can reconstruct N'' from T'' as follows. First add a tree representing $\mathcal{C}|X_r$ below a reticulation and subsequently add x below another reticulation. Notice however that CASS always adds the reticulations below nodes inserted into edges, while in N'' a reticulation might be a child of a node v with indegree one and outdegree at least two (observe that v cannot be a reticulation because $X_r \neq \emptyset$ and that v cannot have indegree 0 because CASS adds a dummy root with an edge to the old root). CASS adds the new reticulation below a node inserted into the edge entering v instead of below v, which leads to a network N''' that also represents $\mathcal{C}$.

To conclude the proof, consider the case $X_r = \emptyset$. Assume without loss of generality that N'' contains one reticulation with indegree 3 (if it contains two reticulations with indegree 2 then we can contract the edge between these reticulations). In this case, CASS constructs a network N''' representing $\mathcal{C}$ from T'' as follows. It first adds a dummy leaf d below a reticulation, then it adds x below another reticulation. This second reticulation is added below nodes inserted into the edge entering d and one other edge. Before outputting the network, CASS removes d and contracts the edges between the two reticulations. As in the previous case, whenever in N'' the reticulation hangs below a node with outdegree at least two, CASS hangs the reticulation below a node inserted into the edge entering v instead. The resulting network N''' represents $\mathcal{C}$. This concludes the proof. □

Lemma 2. CASS runs in time $O(|\mathcal{X}|^{3k+2} \cdot |\mathcal{C}|)$, if k is fixed.

Proof. Let $n = |\mathcal{X}|$ and $m = |\mathcal{C}|$. We analyze the running time of constructing a simple level-$\leq k$ network, since all other computations can clearly be done in $O(n^{3k+2} \cdot m)$ time. We will show by induction on k' that a call to CASS$(\mathcal{C}, \mathcal{X}, k, k')$ takes at most $O(n^{3k'+2} \cdot m)$ time and returns at most $O(n^{3k'})$ networks, for fixed k'. The lemma will follow because in the original call $k' = k$.

For $k' = 0$, CASS only checks if there exists a tree representing $\mathcal{C}$, which can clearly be done in $O(n^2 \cdot m)$ time and leads to at most one network. Suppose $k' \geq 1$. The algorithm loops through $O(n)$ taxa and at most $O(n^{3k'-3})$ recursively created networks. For each network, the algorithm loops through all pairs of edges. For fixed k', each network contains $O(n)$ edges, since a tree contains at most $2n - 2$ edges and the algorithm adds a constant number of edges in each iteration. For each combination of edges, CASS checks if the constructed network represents all clusters. This can be done by looping through the at most $2^{k'}$ ways of switching edges on and off and checking if all m clusters are represented by one of the resulting trees, in $O(2^{k'} \cdot m \cdot n^2)$ time. This is the bottleneck of all computations. Thus, the total time needed by CASS is $O(n \cdot n^{3k'-3} \cdot n^2 \cdot 2^{k'} \cdot m \cdot n^2)$, which is $O(n^{3k'+2} \cdot m)$ for fixed k'. Similarly, the number of constructed networks is at most $O(n \cdot n^{3k'-3} \cdot n^2)$ and hence $O(n^{3k'})$. □

Lemma 3. For each $r \geq 2$, there exists a set $\mathcal{C}_r$ of clusters such that there exists a network with two reticulations that represents $\mathcal{C}_r$ while any galled network representing $\mathcal{C}_r$ contains at least r reticulations.

Proof. Consider a simple level-2 network N of type $2a$, with r leaves on each of the sides B, C and E and a single leaf on side F. Let $\mathcal{C}_r$ be the set of all clusters represented by N. Suppose that there exists a galled network N' representing $\mathcal{C}_r$ and containing $r' < r$ reticulations. It is easy to check that the incompatibility graph of $\mathcal{C}_r$ (excluding singleton clusters) is connected and hence that N' contains just one biconnected component (except for the leaves). Thus, the r' reticulations of N' each have a leaf as child. Let $\mathcal{C}'$ be the

result of removing the r' taxa labeling these leaves from $\mathcal{C}_r$. It follows that there exists a tree representing $\mathcal{C}'$ and hence that $\mathcal{C}'$ is compatible. However, $\mathcal{C}'$ clearly contains at least one leaf on each of the sides B, C and E of N, say a leaf b on side B, c on side C and e on side E. Hence, there will be a cluster $X \in \mathcal{C}'$ containing b and e but not c and a cluster $Y \in \mathcal{C}'$ containing c and e but not b. It follows that X and Y are incompatible, which is a contradiction because we have already shown that $\mathcal{C}'$ is compatible. $\qquad\square$